Quantitative Performance Assessment of Multiobjective Optimizers: The Average Runtime Attainment Function

Dimo Brockhoff, Anne Auger, Nikolaus Hansen, Tea Tušar

Quantitative Performance Assessment of Multiobjective Optimizers: The Average Runtime Attainment Function*

Dimo Brockhoff[1] Anne Auger[1] Nikolaus Hansen[1] Tea Tušar[2]

[1] Inria Saclay – Ile-de-France and CMAP, UMR CNRS 7641, Ecole Polytechnique
firstname.lastname@inria.fr
[2] Department of Intelligent Systems, Jožef Stefan Institute, Slovenia
tea.tusar@ijs.si

Abstract. Numerical benchmarking of multiobjective optimization algorithms is an important task needed to understand and recommend algorithms. So far, two main approaches to assessing algorithm performance have been pursued: using set quality indicators, and the (empirical) attainment function and its higher-order moments as a generalization of empirical cumulative distributions of function values. Both approaches have their advantages but rely on the choice of a quality indicator and/or take into account only the location of the resulting solution sets and not *when* certain regions of the objective space are attained. In this paper, we propose the average runtime attainment function as a quantitative measure of the performance of a multiobjective algorithm. It estimates, for any point in the objective space, the expected runtime to find a solution that weakly dominates this point. After defining the average runtime attainment function and detailing the relation to the (empirical) attainment function, we illustrate how the average runtime attainment function plot displays algorithm performance (and differences in performance) for some algorithms that have been previously run on the biobjective `bbob-biobj` test suite of the COCO platform.

1 Introduction

Performance assessment of black-box algorithms is an important task to *understand* and to *recommend* algorithms in the contexts of practical applications and the design of new algorithms. A lot of progress has been made recently in single-objective optimization on improving standards to assess algorithm performance properly. Particularly, for instance, through the introduction of runtime distributions or data profiles [10, 12], performance profiles [4] or software platforms for automated benchmarking, such as COCO [8]. One important aspect of performance assessment, as advocated within the COCO framework, is the need for *quantitative* performance measures. There exist typically two ways of collecting data within single-objective optimization:

- Record at a given time (budget/function evaluations) the objective function values reached by different runs of an algorithm on a problem. This is referred to as the fixed-budget view (see Fig. 1).

* This is an author version of the EMO 2017 paper published by Springer Verlag. The final publication is available at `www.springerlink.com`.

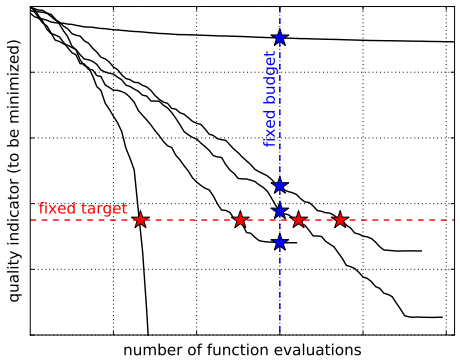


Fig. 1: Fixed-budget versus fixed-target scenarios. Given 5 runs of an algorithm, the fixed budget scenario consists of fixing a cost and recording the objective function values of the 5 runs at this given budget, while the fixed-target scenario consists of fixing a target and recording the number of function evaluations the 5 runs need to reach this target.

– Collect for a certain function value/target the runtime (typically measured in number of function evaluations) to reach this target. In case the target is not reached by a run, one would record the maximal runtime before stopping the run. This is referred to as the fixed-target scenario (see again Fig. 1).

While the first scenario is often argued as close to practice where one has a finite budget to solve a problem, it does not allow for a meaningful *quantitative* performance assessment, because the recorded function values can only be interpreted with the scaling of the objective function in mind (reaching with Algorithm A a function value that is two times smaller than the one reached by Algorithm B could mean either that Algorithm A is marginally faster than B or much faster, depending on the objective function to be optimized). On the contrary, the fixed-target view collects runtimes, which allows for direct quantitative comparisons of the type "Algorithm A is two times faster than Algorithm B (to reach a certain target)". Empirical cumulative distributions (ECDFs) of runtimes collected at a given target—compliant with the fixed-target view and originally introduced as *runtime distributions* [10] and *data profiles* [12]—are now standard to assess performance of single-objective algorithms.

For comparing multiobjective algorithms, the fixed-target view has been adopted only recently, in particular in the context of the COCO framework [8], while the less interpretable fixed-budget view is still by far more common. In both cases, most of the time a quality indicator is used to directly exploit single-objective performance assessment techniques such as statistical tests, box-plots, or data/performance profiles. To be more precise, a single, real-valued quality is assigned to each (set) outcome of an optimization algorithm—either as a quality of a single population, or, as for example in the case of COCO, as the quality of all non-dominated solutions found by an algorithm at an arbitrary point in time. This so-called quality indicator approach to performance assessment is simple but relies on the choice of an indicator (or a set of indicators).

The other well-known approach to assessing the performance of multiobjective algorithms has been proposed in the seminal works by Carlos M. Fonseca and his co-

authors: the visualization and analysis of the (empirical) attainment function [5,6]. The attainment function is thereby a generalization of ECDFs of the best function value at a given time to the multiobjective case and gives, for each point in the objective space, the probability that this point is *attained* (or in other words weakly dominated) by an algorithm at the end of its run. It is typically approximated as the empirical attainment function (EAF) in practice by estimating the probability to attain a point from a (small) set of independent algorithm runs. The EAF can be plotted to get an idea of *where* in objective space an algorithm produces solutions.

However, when investigating the attainment function for a given algorithm, one looses the information on *when* certain points in objective space have been attained by the algorithm. It is the main goal of this paper to propose, based on the idea of the attainment function, a new performance assessment display which allows to investigate also the runtime—the time an algorithm takes to reach certain points in objective space. We thereby transfer the ideas of expected runtime (ERT) and average runtime (aRT) from the single-objective case [7,9] to the multiobjective case through the so-called average runtime attainment (aRTA) function and its associated plot. Similar to the EAF difference plots from [11], we furthermore introduce the aRTA ratio function to compare the average runtimes of two algorithms graphically. Based on a first preliminary implementation of the aRTA and aRTA ratio plots, the performance of a few algorithms from the BBOB 2016 workshop[3], obtained via the COCO platform, is displayed to showcase the usefulness of the new approach.

The paper is organized as follows. Section 2 gives the background on the attainment function approach as well as on the concepts of expected and average runtime, our aRTA functions are based upon. Section 3 details the new displays while Section 4 showcases them for a few data sets, obtained on the `bbob-biobj` test suite of the COCO platform, and gives details on the provided implementation. Finally, Section 5 concludes the paper and discusses the limitations of the proposed performance displays.

2 Preliminaries

Throughout the paper, we consider the minimization of a multiobjective problem with m objective functions defined over a general search space Ω, i.e. we minimize

$$x \in \Omega \mapsto (f_1(x), \ldots, f_m(x)) \in \mathbb{R}^m \tag{1}$$

in which no specific assumption on the search space Ω is made. The search space can actually be discrete, continuous, etc. and in the remainder, we therefore focus our investigations on objective vectors $z \in \mathbb{R}^m$ only and, for simplicity, use the terms *solution* and *objective vector* interchangeably. We denote the coordinates of an objective vector $z \in \mathbb{R}^m$ as $(z_1, \ldots, z_m)$.

The weak dominance relation is defined for two objective vectors y and z of $\mathbb{R}^m$ as y *weakly dominates* z, denoted $y \preceq z$, if and only if $y_i \leq z_i$ for all i. A generalization of the weak dominance relation towards sets of objective vectors is straightforward by defining weak dominance between two sets Y and Z (both subsets of $\mathbb{R}^m$) whenever

[3] see `https://numbbo.github.io/workshops/BBOB-2016/`

for each $x \in Z$ there exists a $y \in Y$ such that y weakly dominates z. In this case, we follow the notation of [6] and write $Y \trianglelefteq Z$.

If in a set of objective vectors A, all pairs of objective vectors are mutually non-dominated (in terms of the above weak dominance relation), we call A a set of mutually non-dominated objective vectors, or also a set of non-dominated vectors or even simpler, a non-dominated set.

2.1 Empirical Attainment Function

Given a set of non-dominated vectors $\mathcal{X} = \{X^1, \ldots, X^p\}$ (of random) size p and given a target vector $z \in \mathbb{R}^m$ of the objective space, we say that the target is reached (or attained) by the vector-set $\mathcal{X}$ if $\mathcal{X} \trianglelefteq z$.

Given N such sets of non-dominated vectors $\{\mathcal{X}_1, \ldots, \mathcal{X}_N\}$ (each containing a random number of non-dominated vectors), the empirical attainment function (EAF) introduced in [6] is defined as

$$\alpha(z) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\{\mathcal{X}_i \trianglelefteq z\} \ . \tag{2}$$

The EAF maps the objective space $\mathbb{R}^m$ to $[0, 1]$. In practice, the EAF is computed for N sets of non-dominated vectors that are the outcome of N independent trials collected at the end of a run or at a fixed budget T. It estimates the probability of an optimizer to find, within the budget T, an objective vector which is at least as good as the target vector z, where "at least as good" is interpreted in the weak dominance sense. Equivalently, the EAF estimates the probability to attain the region

$$\mathcal{A}(z) = \{y \in \mathbb{R}^m | z \preceq y\} \ . \tag{3}$$

To emphasize the dependence on T, we denote by $\alpha_T(z)$ the empirical attainment function of non-dominated vectors collected at a time T. In the case of a single objective ($m = 1$), $z \to \alpha_T(z)$ is the empirical cumulative distribution of the objective functions distribution reached at T [6]. That is, it is the empirical cumulative distribution of the data collected within the fixed budget scenario introduced in the introduction.

What is collected and exploited? To summarize, the empirical attainment function relies solely on N sets, each composed of a random number of non-dominated objective vectors, which have been collected at some point in time T of an algorithm run. In order to allow for meaningful comparisons of algorithms, the time T shall be the same for all algorithms considered. To see the evolution of the algorithm performances during the search, multiple empirical attainment functions (for varying T) have to be displayed.

2.2 Expected Runtime (ERT) and Average Runtime (aRT)

While the EAF assumes a fixed budget, we remind here the definition of the expected runtime and average runtime that assume a fixed-target scenario. Consider the case

where algorithm A either successfully reaches the target value f_{target} or it does not. The ERT [7] corresponds to the expected runtime of a conceptual algorithm that would restart A till obtaining a success, i.e., till f_{target} is reached. Given that algorithm A has a probability of success of p_s, an expected runtime for successful runs of $E[\text{RT}^{\text{s}}]$ and an expected runtime for unsuccessful runs of $E[\text{RT}^{\text{us}}]$, the ERT can be expressed as

$$\text{ERT}(f_{\text{target}}) = \frac{1-p_s}{p_s} E[\text{RT}^{\text{us}}] + E[\text{RT}^{\text{s}}] \ . \tag{4}$$

It allows to compare in a meaningful and quantitative way algorithms that have a small probability of success, but converge fast when they do, with algorithms with a larger probability of success and a slower convergence rate.

An estimator for ERT is the average runtime (aRT, see also for example [7]). Given N runs of an algorithm with N_{s} successes to reach the target f_{target} and an overall number of function evaluations of $\text{FE}(N) = \sum_{i=1}^{N} T_i$ that includes the number of function evaluations for successful and unsuccessful runs, the aRT equals

$$\text{aRT}(f_{\text{target}}) = \frac{\text{FE}(N)}{N_{\text{s}}} \ . \tag{5}$$

The estimator for ERT when all unsuccessful runs have a number of function evaluations equal to a cutoff number was actually first proposed in [9].

3 Average Runtime Attainment Functions

We introduce in this section the average runtime attainment (aRTA) function that can be seen as a generalization of the attainment function where the information on the runtime to reach a target vector is re-introduced.

Similar to the EAF difference plots for comparing the EAFs of two algorithms in [11], we introduce the aRTA ratio function in addition for an easier comparison of the average runtimes between two algorithms.

3.1 Average Runtime Attainment Function

Compliant with the fixed-target approach, we fix a target vector $z \in \mathbb{R}^m$ and collect the minimal number of function evaluations (runtime) $T(z)$ to obtain a solution that weakly dominates z. If a run was not successful, that is, it did not find a solution that weakly dominates z, we collect the runtime of the run when it stopped. We assume that over N trials of the algorithm, we have collected all N runtimes, $T_1(z), \ldots T_N(z)$, and obtained N_{s} successes ($\leq N$). Then, the aRTA is the function defined as

$$\text{aRTA}(z) = \frac{\sum_{i=1}^{N} T_i(z)}{N_{\text{s}}} \ . \tag{6}$$

Comparing (5) with (6), we see that aRTA is the natural generalization of the aRT estimator used in the single-objective case where we have adapted the notion of success from reaching a function value below a certain target to reaching a solution that weakly

dominates a target vector. The aRTA function maps $\mathbb{R}^m$ to $\mathbb{R}^+$. Note in particular that, like in the single-objective case, the maximum number of function evaluations recorded for an algorithm effects the aRT values which Section 4.1 investigates in more detail.

In order to plot $\mathrm{aRTA}(z)$ in practice, the average runtime values of $\mathbb{R}^+$ need to be mapped to a color as we will showcase in the following section. Before, however, let us transfer another known concept around empirical attainment functions.

3.2 Average Runtime Attainment Ratio Function

In order to compare the aRTs of two algorithms more easily, we advocate to display the plots of the so-called aRTA ratio function, similar to the EAF difference plots of [11]. [4]

To compare the aRTA functions of algorithms A and B, we can, in principle, plot the ratio of the two aRTA function values for both algorithms and each objective vector directly, i.e. we can plot

$$\mathrm{aRTA}_{\mathrm{ratio}}(z) = \frac{\mathrm{aRTA}^{\mathrm{B}}(z)}{\mathrm{aRTA}^{\mathrm{A}}(z)} \tag{7}$$

as long as $\mathrm{aRTA}_{\mathrm{ratio}}(z)$ is well defined (it is not well defined as soon as one or both of the aRTA function values are not finite). Since the measured runtimes are comparable on a ratio scale with a non-arbitrary zero, we prefer aRTA *ratios* here over *differences* like in the EAF case [11]. This has the immediate effect that aRTA ratio functions are interpretable without the need to know any absolute values. To have an easier-to-read plot and to also cope with undefined $\mathrm{aRTA}_{\mathrm{ratio}}$ values, we actually propose to display a slight variant of the above.

If $\mathrm{aRTA}_{\mathrm{ratio}}(z)$ is well-defined and larger than 1, indicating an advantage for algorithm A, we simply plot $\mathrm{aRTA}_{\mathrm{ratio}}(z)$. Likewise, if $\mathrm{aRTA}_{\mathrm{ratio}}(z)$ is well-defined and smaller than 1, indicating an advantage for algorithm B, we plot $\frac{1}{\mathrm{aRTA}_{\mathrm{ratio}}(z)}$, color-coded with a different colormap instead—making it possible to easily compare advantages of algorithm A with advantages of algorithm B in the sense of statements like "Algorithm A is X times better than algorithm B in attaining the objective vector z". Undefined values of $\mathrm{aRTA}_{\mathrm{ratio}}(z)$, where only one algorithm possesses a finite aRT value (because for the other, all runs are unsuccessful), can nevertheless be plotted in a color that indicates the algorithm with the more favorable behavior.

3.3 What is collected and exploited?

In comparison to the empirical attainment function, the aRTA and aRTA ratio functions rely on additional information about algorithm runs. In particular, each solution, which is not dominated by already evaluated solutions, needs to be recorded together with the runtime (in number of function evaluations) when it was evaluated by the algorithm. The input to the aRTA function and the aRTA ratio is therefore a sorted list of (number of function evaluations, objective vector) pairs such that for each algorithm run/problem instance, at each point in time, the current (external) archive of non-dominated solutions found so far can be reconstructed from the data.

[4] We opt for displaying ratios here instead of differences as the ratio scale is more natural for statements on runtimes and also has stronger theoretical properties than the interval scale [13].

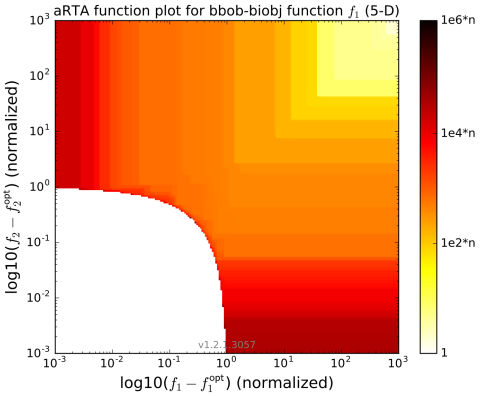


Fig. 2: The aRTA plot for a single algorithm run reduces to a visualization of all recorded non-dominated solutions. The example plot above shows a single run of the algorithm SMS-EMOA with polynomial mutation and SBX crossover on the `bbob-biobj` function 1 (sphere - sphere function) with 5 variables.

Note that in the case of a single recorded run, the aRTA function plot is a visualization of all solutions over time and can be seen as a generalization of a single-objective convergence graph to the multiobjective case, see Fig. 2 for an example.

4 Numerical Examples from COCO

In order to showcase the usefulness of the proposed aRTA plots, we implemented a (preliminary) visualization in Python, which is made available on GitHub[5] and is able to display the algorithm performance from the archive of non-dominated solutions, recorded by the COCO platform [8] on functions from the `bbob-biobj` test suite [14].

In particular, the provided source code reads in the algorithm data from a COCO archive folder in the form of objective vector and runtime pairs (i.e. the function evaluation counter when the objective vector was produced) for each of 10 problem instances per function and dimension n. The code then computes and displays the aRTA function values to weakly dominate for the first time a given objective vector z. When displaying the aRTA ratio function, the data of two algorithms is read in and the aRTA ratios are computed after the calculations of the single aRTA values for both algorithms. In both cases, we use a regular grid of objective vectors z for which we compute the aRT values instead of computing the aRTA areas of constant value exactly. Areas in between grid points are then colored according to the aRTA value (and aRTA ratio respectively) of its lower left corner. All objective values are normalized so that the ideal point is at $[0, 0]$ and the nadir point is at $[1, 1]$. [6]

[5] `https://github.com/numbbo/coco/tree/master/code-postprocessing/aRTAplots`

[6] Note that such a normalization allows for objective values to be larger than 1 and that our plots clips the display to objective values smaller than 10.

All aRTA function plots shown in this paper are in log-scale and, if not specified differently, use a grid of 200×200 points, chosen equidistant on the log-scale between the ideal point $[0, 0]$ and the point $[10, 10]$. [7] The color-coding of the aRTA values is done in a log-scale as well so that the same color ranges are used, for example, for the first 100 function evaluations ("white to yellow") and the function evaluations $10^4 n, \ldots, 10^6 n$ ("red to black"). Note that the color scheme is absolute to allow for comparisons across figures and all solutions produced beyond the maximal budget of $10^6 n$ function evaluations are not used in the display. Although from all data sets submitted to the BBOB-2016 workshop only the one of HMO-CMA-ES contains solutions beyond this threshold, Section 4.1 investigates the influence of this parameter on the aRTA function plots in detail.

In order to cope with the large data sets produced by the COCO platform[8], our implementation removes all but one of the recorded solutions within each grid cell before the computation of the aRT values. Thereby, the solution with the smallest function evaluation count per instance and grid cell is kept to not alter the plots while downsampling. As we will see later on in Section 4.2, this *downsampling* significantly reduces the computation time for the aRTA function plots. This section closes with showing a few examples of algorithm comparisons in Section 4.3.

All experiments for this paper have been run with COCO, version 1.2.1—more precisely with the code of the feature-branch as of commit `1c22851` on Dec. 31, 2016.

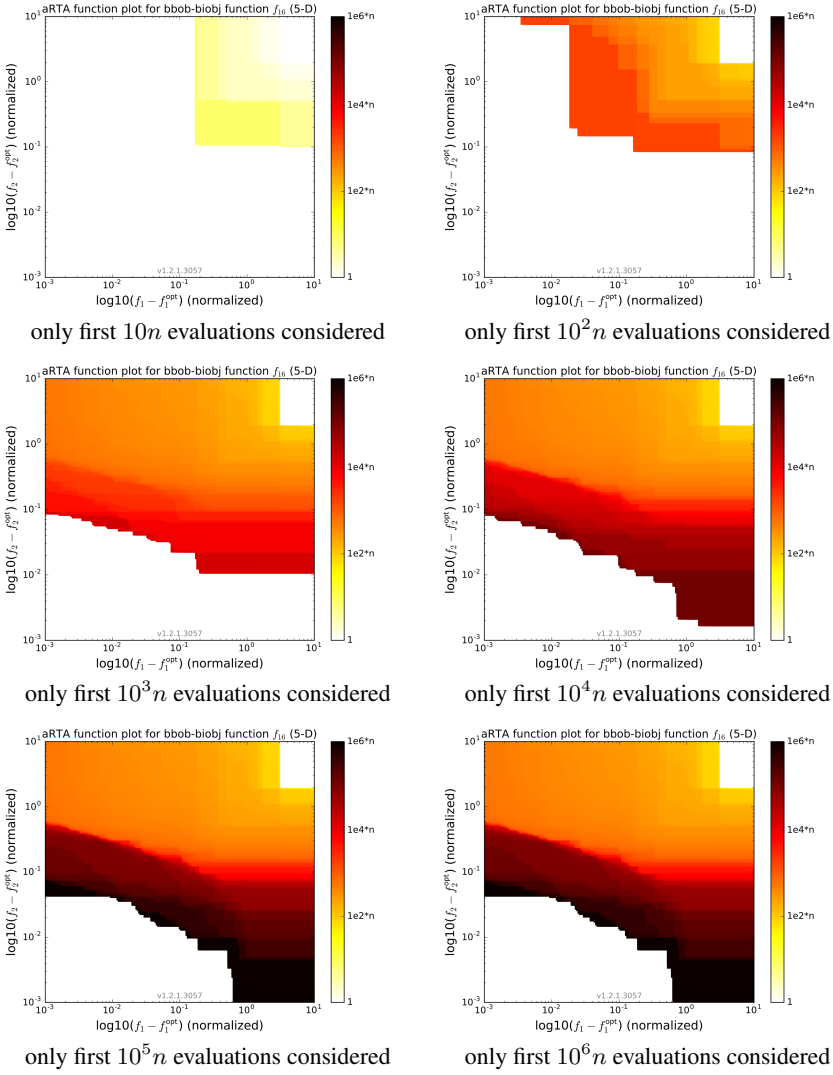
4.1 The Influence of the Maximal Budget of Function Evaluations

As a first investigation of the aRTA function plots we consider the influence of the maximal budget parameter, which specifies a threshold for function evaluations after which no solution is taken into account anymore. While it is typically not needed to change this parameter from its default setting of $10^6 n$ function evaluations to display the available COCO data, Fig. 3 shows the influence of the maximal budget on the aRTA function plots for the Matlab implementation of NSGA-II on the 5-dimensional separable ellipsoid - Rastrigin problem (function f_{16} in the `bbob-biobj` test suite).

Two observations can be made from Fig. 3. First, we see that a larger maximal budget value (and thus data from longer runs) results in a larger range of aRTA values, which are distributed in a larger area of the objective space. Secondly, increasing the maximal budget can change the color of those areas in the objective space that have not been attained in all runs at the lower budget value. The color can get darker (see the difference for maximal budget set to $10n$ and $10^2 n$) or lighter (see the difference on the upper left part of the plot for maximal budget set to $10^2 n$ and $10^3 n$), depending on which change in the aRTA fraction (the increasing runtimes in the numerator or the increasing success rate in the denominator) has a larger effect. Once an area of the objective space has been attained in all runs, its color cannot change any longer.

[7] Note that with the `logscale` parameter in the provided source code, the log-scale can be easily turned on and off.

[8] A single function/dimension combination with 10 instances produces up to 930MB of data.



only first $10n$ evaluations considered | only first $10^2 n$ evaluations considered

only first $10^3 n$ evaluations considered | only first $10^4 n$ evaluations considered

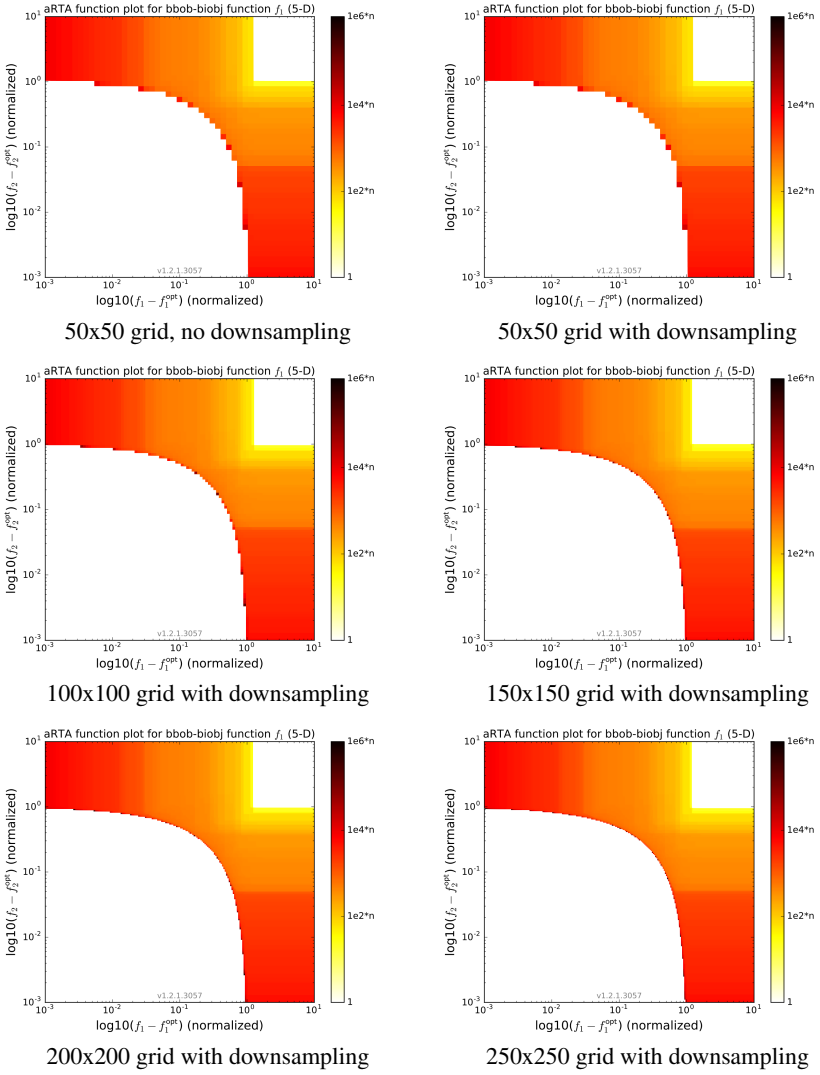
only first $10^5 n$ evaluations considered | only first $10^6 n$ evaluations considered

Fig. 3: Influence of the maximal number of function evaluations considered on the average runtime plots for GA-MULTIOBJ-NSGA-II on function f_{16} (separable ellipsoid - Rastrigin) in dimension 5. Shown are, from top left to bottom right, the aRTA function plots when only the first $10n$, $10^2 n$, $10^3 n$, $10^4 n$, $10^5 n$, and $10^6 n$ function evaluations are considered.

4.2 The Influence of Downsampling Data and Different Grid Sizes

To investigate the influence of the possible downsampling on the aRTA plots as well as on the time it takes to produce them[9], we use the data from the Matlab implementation

[9] All experiments were performed on an Intel Core i7-5600U CPU Windows 7 laptop with 8GB of RAM.



50x50 grid, no downsampling | 50x50 grid with downsampling

100x100 grid with downsampling | 150x150 grid with downsampling

200x200 grid with downsampling | 250x250 grid with downsampling

Fig. 4: First row: Downsampling the input data does not change the plot if per grid cell the solution with the lowest function evaluations count is kept. All other plots show the influence of the grid size on the aRTA plots, from a 50x50 grid, which takes about 6 seconds to produce, up to a 250x250 grid, which takes about 6.5 minutes to produce.

of NSGA-II, as submitted to the BBOB-2016 workshop, on the example of the f_1 function (sphere - sphere). Without downsampling, the 10 instances of the data have 6755 to 9710 non-dominated solutions per instance (78,318 solutions in total), and it takes about 29 minutes to produce the single aRTA plot with 200×200 gridpoints, see Fig. 4. The provided source code is certainly not optimized for speed, but a runtime to produce a single aRTA plot of about half an hour is, of course, not acceptable in practice. With

downsampling, i.e. taking only into account a single solution per grid cell, the time to produce the same plot can be reduced significantly. In order to further decrease the time to produce a single aRTA plot, we can trade the runtime with accuracy and change the grid size. This results in the following runtimes:

- ca. 6.5 minutes for a 250x250 grid (826–1062 solutions per instance[10]), down from about 49 minutes without downsampling,
- ca. 3.5 minutes for a 200x200 grid (685–905 solutions per instance), down from about 29 minutes without downsampling,
- ca. 1.5 minutes for a 150x150 grid (523–698 solutions per instance), down from about 15 minutes without downsampling
- ca. 32 seconds for a 100x100 grid (370–483 solutions per instance), down from about 7 minutes without downsampling, and to
- ca. 6 seconds for a 50x50 grid (173–244 solutions per instance), down from about 1.5 minutes without downsampling

When comparing the actual aRTA plots of Fig. 4 for the different grid sizes, we observe that increasing the number of grid cells increases the accuracy while an increase from the 200x200 to the 250x250 grid is hardly visible. As a good trade-off between accuracy and time to produce the plots, we therefore recommend using downsampling and the 200x200 grid as default, which is used for all plots in the remainder of the paper.

4.3 A Few Examples of Algorithm Comparisons with aRTA Function Plots

In this last section, we investigate some of the data, submitted to the BBOB 2016 workshop, for which participants were asked to benchmark their favorite algorithm on the `bbob-biobj` test suite via the COCO platform. The plots of the aRTA in Figs. 6, 7, and 8 can be seen as supplements to the empirical cumulative distribution functions (ECDFs), provided by the COCO platform by default. Fig. 5 for example shows the ECDF of the runtimes for the algorithms RS-5, MO-DIRECT-hv-rank, GA-MULTIOBJ-NSGA-II, and SMS-EMOA-PM to reach 58 target hypervolume indicator values on the 5-dimensional sphere - sphere problem (f_1). RS-5 is thereby a simple random search within the domain $[-5, 5]$ [2], MO-DIRECT-hv-rank is an extension of the DIviding RECTangles approach to multiobjective optimization [15], GA-MULTIOBJ-NSGA-II is the default Matlab implementation of the standard NSGA-II [1], and SMS-EMOA-PM is the standard SMS-EMOA variant with polynomial mutation and SBX crossover [3]. The random search will be used here as a reference algorithm to which the other three algorithms are compared to.

If we look carefully at Fig. 5, we see that for this particular problem, MO-DIRECT-hv-rank is at all times better than RS-5, SMS-EMOA-PM is worse in the beginning (because it initializes its population in the much larger space $[-100, 100]$) and better in the end compared to RS-5, and finally the NSGA-II is better in the very beginning, then worse, and finally better again than RS-5. Figures 6, 7, and 8 show the corresponding

[10] Note that it is not necessarily the case that the instance with the smallest (largest) number of solutions recorded results in the smallest (largest) set of downsampled points.

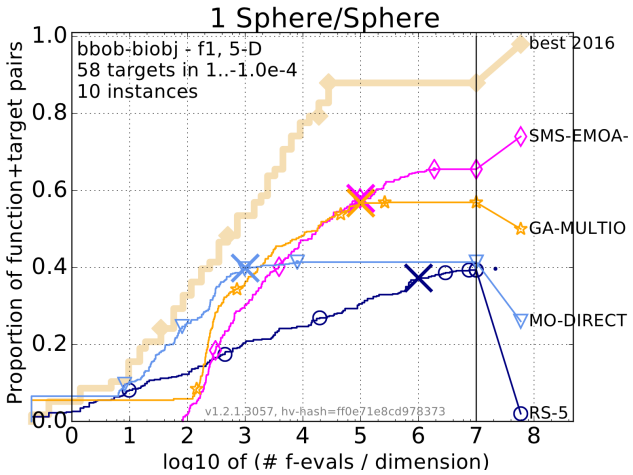


Fig. 5: Empirical cumulative distribution function of the runtimes of four algorithms to reach 58 target values on the `bbob-biobj` function f_1 (sphere - sphere) in dimension 5.

aRTA plots for the single algorithms as well as the corresponding aRTA ratio plots—displaying the same trends of *when* an algorithm is better than another and in addition also *where* in objective space and by *how much*.

A large difference between the shown algorithms lie in particular in their different initialization strategies. While the random search samples always uniformly at random in the set $[-5, 5]^n$ with n being the search space dimension, SMS-EMOA-PM samples its initial population from the much larger space $[-100, 100]^n$. The NSGA-II variant, displayed here, samples all but the first solution in its initial population also from $[-100, 100]^n$ and the first solution according to an isotropic Gaussian distribution around the search space origin. MO-DIRECT-hv-rank, finally, evaluates the search space origin as first solution. These different initialization strategies have a large impact on the algorithm performance during the first evaluations and beyond, which can be seen both in the ECDFs of Fig. 5 and the aRTA function plots. For the larger budgets and therefore areas close to the Pareto front in the aRTA (ratio) function plots, the initialization strategy seems to have no influence anymore and it is the algorithm's ability to approximate the Pareto front well which plays the biggest role in both the ECDFs and the aRTA plots.

5 Conclusion

We have proposed the average runtime attainment (aRTA) function as an alternative to the empirical attainment function to evaluate performance of multiobjective optimizers. In contrast to the latter, the aRTA function displays *quantitative* measurements of *when* the region that weakly dominates an objective vector z was reached for the first time. We have illustrated a simple display of the aRTA function (and of the aRTA ratio function for comparing two algorithms) on a grid using some data from the COCO platform.

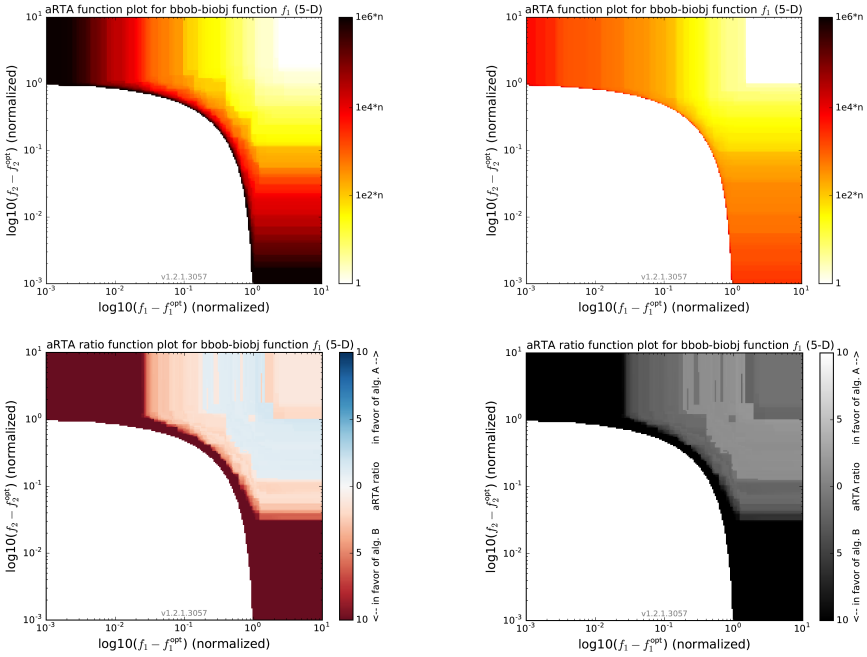


Fig. 6: Average runtime plots for RS-5 (Algorithm A, top left) and MO-DIRECT-hv-rank (Algorithm B, top right) together with the corresponding aRTA ratio plot (bottom row, left: colored, right: optimized for grayscale) on the 5-dimensional `bbob-biobj` function f_1.

Two shortcomings of the current implementation must be mentioned. The running time of the code that produces exact plots on an objective space grid is relatively high for practical purposes (in the order of minutes) even when the number of input solutions is downsampled to a single solution per grid cell and per instance. In addition, the displayed data, resulting from the COCO platform, contains results from 10 different instances of the same problem with potential discrepancies in the objective space. Here, the proposed aRTA function would be even more useful (and interpretable) if applied to data from independent runs on the same problem instance.

Last, let us discuss the generalization of the aRTA function displays to a higher number of objectives. While their definition is not restricted to two objective functions and their computation on a similar grid in higher dimension is possible with the same computational complexity per grid point, the aRTA function cannot be practically displayed in the same way as for two-objective problems: already for a three-dimensional grid, any display can only show 2-dimensional cuts through the grid or the surfaces of all points with a certain, predefined aRTA function value. We therefore expect the aRTA function to be less informative in higher dimensions than for two-objective problems as showcased here.

Acknowledgments: The authors acknowledge the support of the French National Research Agency (ANR) within the Modèles Numérique project "NumBBO – Analysis, Improvement and Evaluation of Numerical Blackbox Optimizers" (ANR-12-MONU-

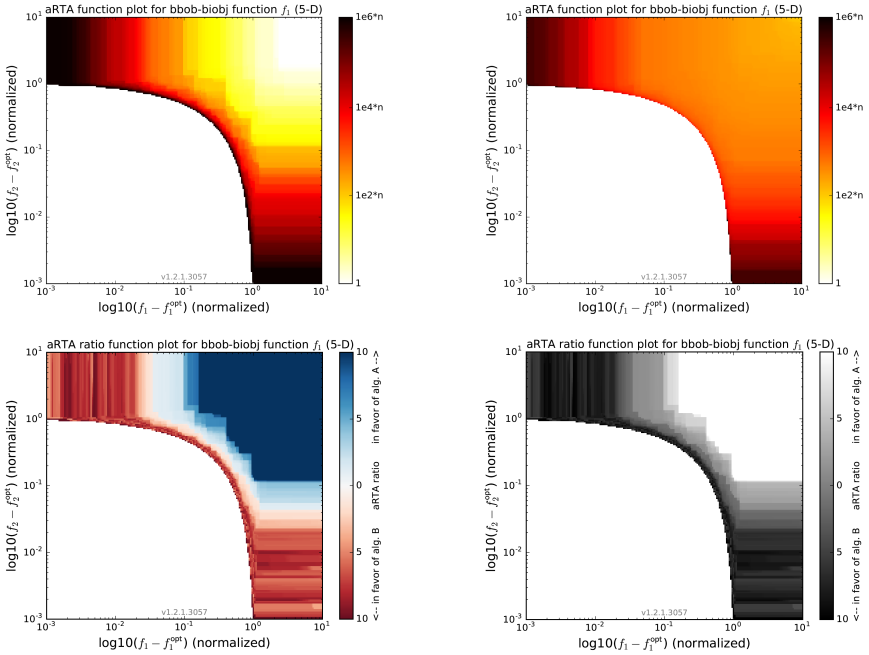


Fig. 7: Average runtime plots for RS-5 (Algorithm A, top left) and SMS-EMOA-PM (Algorithm B, top right) together with the corresponding aRTA ratio plot (bottom row, left: colored, right: optimized for grayscale) on the 5-dimensional `bbob-biobj` function f_1.

0009). In addition, this work is part of a project that has received funding from the *European Union's Horizon 2020 research and innovation program* under grant agreement No. 692286. This work was partially funded also by the Slovenian Research Agency under research program P2-0209. We finally thank the anonymous reviewers for their valuable comments.

References

1. Auger, A., Brockhoff, D., Hansen, N., Tušar, D., Tušar, T., Wagner, T.: Benchmarking MATLAB's gamultiobj (NSGA-II) on the Bi-objective BBOB-2016 Test Suite. In: GECCO (Companion) workshop on Black-Box Optimization Benchmarking (BBOB'2016). pp. 1233–1239. ACM (2016)
2. Auger, A., Brockhoff, D., Hansen, N., Tušar, D., Tušar, T., Wagner, T.: Benchmarking the Pure Random Search on the Bi-Objective BBOB-2016 Testbed. In: GECCO (Companion) workshop on Black-Box Optimization Benchmarking (BBOB'2016). pp. 1217–1223. ACM (2016)
3. Auger, A., Brockhoff, D., Hansen, N., Tušar, D., Tušar, T., Wagner, T.: The Impact of Variation Operators on the Performance of SMS-EMOA on the Bi-objective BBOB-2016 Test Suite. In: GECCO (Companion) workshop on Black-Box Optimization Benchmarking (BBOB'2016). pp. 1225–1232. ACM (2016)
4. Dolan, E.D., Moré, J.J.: Benchmarking Optimization software with Performance Profiles. Mathematical Programming 91, 201–213 (2002)

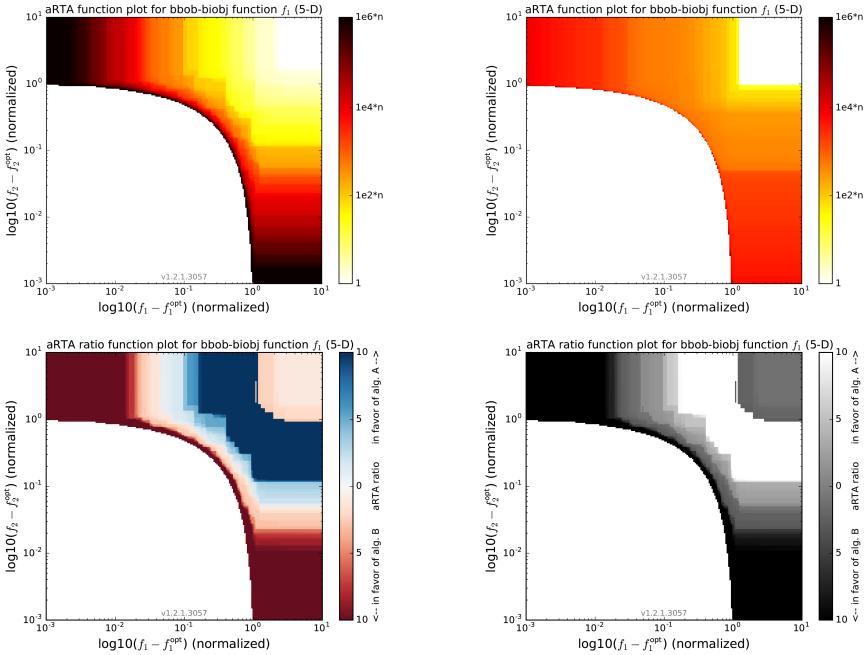


Fig. 8: Average runtime plots for RS-5 (Algorithm A, top left) and GA-MULTIOBJ-NSGA-II (Algorithm B, top right) together with the corresponding aRTA ratio plot (bottom row, left: colored, right: optimized for grayscale) on the 5-dimensional `bbob-biobj` function f_1.

5. Fonseca, C.M., Fleming, P.J.: On the Performance Assessment and Comparison of Stochastic Multiobjective Optimizers. In: Parallel Problem Solving from Nature (PPSN IV). LNCS, vol. 1141, pp. 584–593. Springer, Berlin, Germany (1996)
6. Grunert da Fonseca, V., Fonseca, C.M., Hall, A.O.: Inferential Performance Assessment of Stochastic Optimisers and the Attainment Function. In: Zitzler, E., et al. (eds.) Conference on Evolutionary Multi-Criterion Optimization (EMO 2001). LNCS, vol. 1993, pp. 213–225. Springer (2001)
7. Hansen, N., Auger, A., Brockhoff, D., Tušar, D., Tušar, T.: Coco: Performance assessment. CoRR abs/1605.03560 (2016), `http://arxiv.org/abs/1605.03560`
8. Hansen, N., Auger, A., Mersmann, O., Tušar, T., Brockhoff, D.: COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. CoRR abs/1603.08785 (2016), `http://arxiv.org/abs/1603.08785`
9. Hoos, H., Stützle, T.: Evaluating Las Vegas Algorithms: Pitfalls and Remedies. In: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence. pp. 238–245. Morgan Kaufmann Publishers Inc. (1998)
10. Hoos, H.H., Stützle, T.: Stochastic local search: Foundations & applications. Elsevier (2004)
11. López-Ibáñez, M., Paquete, L., Stützle, T.: Exploratory Analysis of Stochastic Local Search Algorithms in Biobjective Optimization. In: Experimental Methods for the Analysis of Optimization Algorithms, chap. 9, pp. 209–222. Springer (2010)
12. Moré, J., Wild, S.: Benchmarking Derivative-Free Optimization Algorithms. SIAM J. Optimization 20(1), 172–191 (2009), preprint available as Mathematics and Computer Science Division, Argonne National Laboratory, Preprint ANL/MCS-P1471-1207, May 2008.
13. Stevens, S.S.: On the theory of scales of measurement. Science 103(2684), 677–680 (1946)

14. Tušar, T., Brockhoff, D., Hansen, N., Auger, A.: COCO: the bi-objective black box optimization benchmarking (bbob-biobj) test suite. CoRR abs/1604.00359 (2016), http://arxiv.org/abs/1604.00359
15. Wong, C., Al-Dujaili, A., Sundaram, S.: Hypervolume-Based DIRECT for Multi-Objective Optimisation. In: GECCO (Companion) workshop on Black-Box Optimization Benchmarking (BBOB'2016). pp. 1201–1208. ACM (2016)